

DESIGN OF AUTOMATICALLY ADAPTABLE WEB WRAPPERS

Emilio Ferrara

Department of Mathematics, University of Messina, Italy
emilio.ferrara@unime.it

Robert Baumgartner

Lixto Software GmbH, Vienna, Austria
robert.baumgartner@lixto.com

Keywords: Semantic Web, Information Extraction, Data Mining

Abstract: Nowadays, the huge amount of information distributed through the Web motivates studying techniques to be adopted in order to extract relevant data in an efficient and reliable way. Both academia and enterprises developed several approaches of Web data extraction, for example using techniques of artificial intelligence or machine learning. Some commonly adopted procedures, namely *wrappers*, ensure a high degree of precision of information extracted from Web pages, and, at the same time, have to prove robustness in order not to compromise quality and reliability of data themselves.

In this paper we focus on some experimental aspects related to the robustness of the data extraction process and the possibility of automatically adapting wrappers. We discuss the implementation of algorithms for finding similarities between two different version of a Web page, in order to handle modifications, avoiding the failure of data extraction tasks and ensuring reliability of information extracted. Our purpose is to evaluate performances, advantages and draw-backs of our novel system of *automatic wrapper adaptation*.

1 INTRODUCTION

The World Wide Web today contains an exterminated amount of information, mostly unstructured, under the form of Web pages, but also documents of various nature. During last years big efforts have been conducted to develop techniques of information extraction on top of the Web. Approaches adopted spread in several fields of Mathematics and Computer Science, including, for example, logic-programming and machine learning. Several projects, initially developed in academic settings, evolved in commercial products, and it is possible to identify different methodologies to face the problem of Web data extraction. A widely adopted approach is to define *Web wrappers*, procedures relying on analyzing the structure of HTML Web pages (i.e. DOM tree) to extract required information. Wrappers can be defined in several ways, e.g. most advanced tools let users to design them in a visual way, for example selecting elements of interest in a Web page and defining rules for their extraction and validation, semi-automatically; regardless of their generation process, wrappers intrinsically refer

to the HTML structure of the Web page at the time of their creation. Thus, introducing not negligible problems of robustness, wrappers could fail in their tasks of data extraction if the underlying structure of the Web page changes, also slightly. Moreover, it could happens that the process of extraction does not fail but extracted data are corrupted.

All these aspects clarify the following scenarios: during their definition, wrappers should be as much elastic as possible, in order to intrinsically handle minor modifications on the structure of Web pages (this kind of small local changes are much more frequent than heavy modifications); although elastic wrappers could efficiently react to minor changes, maintenance is required for the whole wrapper life-cycle. Wrapper maintenance is expensive because it requires highly qualified personnel, specialized in defining wrappers, to spend their time in rebuilding or fixing wrappers whenever they stop working properly. For improving this aspect, several commercial tools include notification features, reporting warnings or errors during wrappers execution. Moreover, to increase their reliability, data extracted by wrappers could be subject to

validation processes, and also data cleaning is a fundamental step; some tools provide caching services to store the last working copy of Web pages involved in data extraction processes. Sometimes, it is even more convenient to rewrite *ex novo* a wrapper, instead of trying to find causes of malfunctioning and fixing them, because debugging wrapper executions can be not trivial. The unpredictability of what changes will occur in a specific Web page and, consequently, the impossibility to establish when a wrappers will stop working properly, requires a smart approach to wrapper maintenance.

Our purpose in this paper is to describe the realization and to investigate performances of an automatic process of wrapper adaptation to structural modifications of Web pages. We designed and implemented a system relying on the possibility of storing, during the wrapper definition step, a *snapshot* of the DOM tree of the original Web page, namely a *tree-gram*. If, during the wrapper execution, problems occur, this sample is compared to the new DOM structure, finding similarities on trees and sub-trees, to automatically try adapting the wrapper with a custom degree of accuracy. Briefly, the paper is structured as follows: Section 2 focuses on related work, covering the literature about wrapper generation and adaptation. In Section 3 we explain some concepts related to the tree similarity algorithm implemented, to prove the correctness of our approach. Section 4 shows details about our implementation of the automatic wrapper adaptation. Most important results, obtained by our experimentation, are reported in Section 5. Finally, Section 6 concludes providing some remarks for future work.

2 RELATED WORK

The concept of analyzing similarities between trees, widely adopted in this work, was introduced by Tai (Tai, 1979); he defined the notion of *distance* between two trees as the measure of the dissimilarity between them. The problem of transforming trees into other similar trees, namely *tree edit distance*, can be solved applying elementary transformations to nodes, step-by-step. The minimum cost for this operation represents the tree edit distance between the two trees. This technique shows high computational requirements and complex implementations (Bille, 2005), and do not represents the optimal solution to our problem of finding similarities between two trees. The *simple tree matching* technique (Selkow, 1977) represents a turning point: it is a light-weight recursive top-down algorithm which evaluates position of nodes to measure the degree of isomorphism between two

trees, analyzing and comparing their sub-trees. Several improvements to this technique have been suggested: Ferrara and Baumgartner (Ferrara and Baumgartner, 2011), extending the concept of weights introduced by Yang (Yang, 1991), developed a variant of this algorithm with the capability of discovering clusters of similar sub-trees. An interesting evaluation of the simple tree matching and its weighed version, brought by Kim et al. (Kim et al., 2008), was performed exploiting these two algorithms for extracting information from HTML Web pages; we found their achievements very useful to develop automatically adaptable wrappers.

Web data extraction and adaptation rely especially on algorithms working with DOM trees. Related work, in particular regarding Web wrappers and their maintenance, is intricate: Laender et al. (Laender et al., 2002) presented a taxonomy of wrapper generation methodologies, while Ferrara et al. (Ferrara et al., 2010) discussed a comprehensive survey about techniques and fields of application of Web data extraction and adaptation. Some novel wrapper adaptation techniques have been introduced during last years: a valid hybrid approach, mixing logic-based and grammar rules, has been presented by Chidlovskii (Chidlovskii, 2001). Also machine-learning techniques have been investigated, e.g. Lerman et al. (Lerman et al., 2003) exploited their know-how in this field to develop a system for wrapper verification and re-induction. Meng et al. (Meng et al., 2003) developed the SG-WRAM (Schema-Guided WRapper Maintenance), for wrapper maintenance, starting from the observation that, changes in Web pages, even substantial, always preserve syntactic features (i.e. syntactic characteristics of data items like data patterns, string lengths, etc.), hyperlinks and annotations (e.g. descriptive information representing the semantic meaning of a piece of information in its context). This system has been implemented in their Web data extraction platform: wrappers are defined providing both HTML Web pages and their XML schemes, describing a mappings between them. When the system executes the wrapper, data are extracted under the XML format reflecting the previously specified XML Schema; the wrapper maintainer verifies any issue and, eventually, provides protocols for the automatic adaptation of the problematic wrapper. The XML Schema is a DTD (Document Type Definition) while the HTML Web page is represented by its DOM tree. The framework described by Wong and Lam (Wong and Lam, 2005) performs the adaptation of wrappers previously learned, applying them to Web pages never seen before; they assert that this platform is also able to discover, eventually, new attributes in

the Web page, using a probabilistic approach, exploiting the extraction knowledge acquired through previous wrapping tasks. Also Raposo et al. (Raposo et al., 2005) suggested the possibility of exploiting previously acquired information, e.g. results of queries, to ensure a reliable degree of accuracy during the wrapper adaptation process. Concluding, Kowalkiewicz et al. (Kowalkiewicz et al., 2006) investigate the possibility of increasing the robustness of wrappers based on the identification of HTML elements, inside Web pages, through their XPath, adopting relative XPath, instead of absolute ones.

3 MATCHING UP HTML TREES

Our idea of automatic adaptation of wrappers can be explained as follows: first of all, outlining how to extract information from Web pages (i.e. in our case, how a Web wrapper works); then, describing how it is possible to recover information previously extracted from a different Web page (i.e. how to compare structural information between the two versions of the Web page, finding similarities); finally, defining how to automatize this process (i.e. how to build reliable, robust automatically adaptable wrappers).

Our solution has been implemented in a commercial product¹; Baumgartner et al. (Baumgartner et al., 2009) described details about its design. This platform provides tools to design Web wrappers in a visual way, selecting elements to be extracted from Web pages. During the wrapper execution, selected elements, identified through their XPath(s) in the DOM tree of the Web page, are automatically extracted. Although the wrapper design process lets users to define several restricting or generalizing conditions to build wrappers as much elastic as possible, wrappers are strictly interconnected with the structure of the Web page on top of they are built. Usually, also slight modifications to this structure could alter the wrapper execution or corrupt extracted data.

In this section we discuss some theoretical foundations on which our solution relies; in details, we show an efficient algorithm to find similar elements within different Web pages.

3.1 Methodology

A simple measure of similarity between two trees, once defined their comparable elements, can be established applying the *simple tree matching* algorithm

(Selkow, 1977), introduced in Section 2. We define *comparable elements* among HTML Web pages, nodes, representing HTML elements (or, otherwise, free text) identified by tags, belonging to the DOM tree of these pages. Similarly, we intend for *comparable attributes* all the attributes, both generic (e.g. *class*, *id*, etc.) and type-specific (e.g. *href* for anchor texts, *src* for images, etc.), shown by HTML elements; it is possible to exploit these properties to introduce additional comparisons to refine the similarity measure. Several implementations of the *simple tree matching* have been proposed; our solution exploits an improved version, namely *clustered tree matching* (Ferrara et al., 2010), designed to match up HTML trees, identifying clusters of sub-trees with similar structures, satisfying a custom degree of accuracy.

3.2 Tree Matching Algorithms

Previous studies proved the effectiveness of the *simple tree matching* algorithm applied to Web data extraction tasks (Kim et al., 2008; Zhai and Liu, 2005); it measures the similarity degree between two HTML trees, producing the maximum matching through dynamic programming, ensuring an acceptable compromise between precision and recall.

As improvement to this algorithm, this is a possible implementation of *clustered tree matching*: let $d(n)$ to be the degree of a node n (i.e. the number of first-level children); let $T(i)$ to be the i -th sub-tree of the tree rooted at node T ; let $t(n)$ to be the number of total siblings of a node n including itself.

Algorithm 1 ClusteredTreeMatching(T' , T'')

```

1: if  $T'$  has the same label of  $T''$  then
2:    $m \leftarrow d(T')$ 
3:    $n \leftarrow d(T'')$ 
4:   for  $i = 0$  to  $m$  do
5:      $M[i][0] \leftarrow 0$ ;
6:     for  $j = 0$  to  $n$  do
7:        $M[0][j] \leftarrow 0$ ;
8:     for all  $i$  such that  $1 \leq i \leq m$  do
9:       for all  $j$  such that  $1 \leq j \leq n$  do
10:         $M[i][j] \leftarrow \text{Max}(M[i][j-1], M[i-1][j],$ 
            $M[i-1][j-1] + W[i][j])$  where  $W[i][j] =$ 
            $\text{ClusteredTreeMatching}(T'(i-1), T''(j-1))$ 
11:   if  $m > 0$  AND  $n > 0$  then
12:     return  $M[m][n] * 1 / \text{Max}(t(T'), t(T''))$ 
13:   else
14:     return  $M[m][n] + 1 / \text{Max}(t(T'), t(T''))$ 
15:   else
16:     return 0

```

¹Lixto Suite, www.lixtto.com

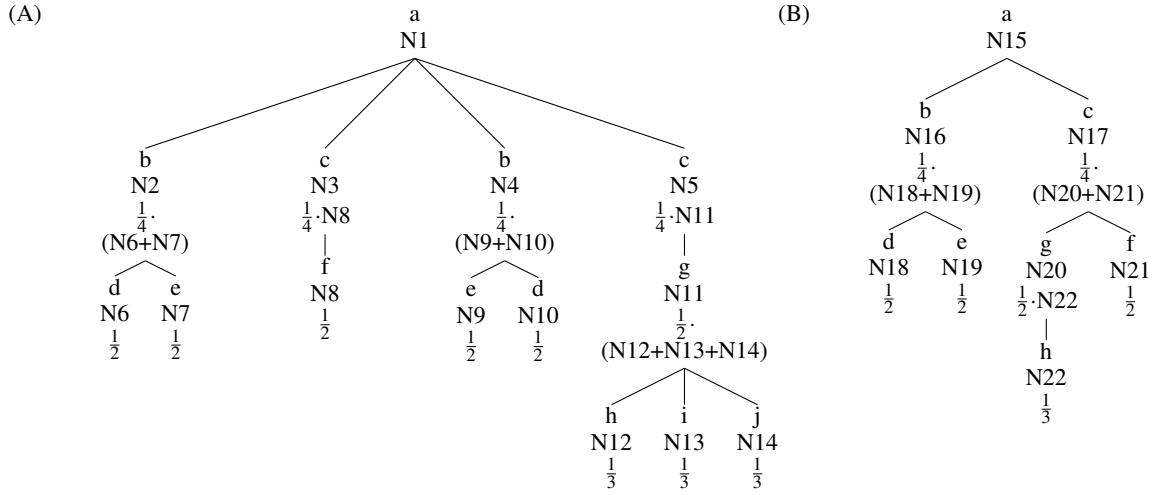


Figure 1: Two labeled trees, A and B, which show similarities in their structures.

The main difference between the *simple* and the *clustered* tree matching is the way of assigning values to matching elements. The first, adopts a fixed matching value of 1; the latter, instead, computes some additional information, retrieved in the sub-trees of matched nodes.

Omitting detail, provided in (Ferrara et al., 2010), the *clustered tree matching* algorithm assigns a weighted value equal to 1, divided by the greater number of siblings, computed between the two compared nodes (also considering themselves).

Figure 1 shows two similar simple rooted, labeled trees, and the way of assignment of weights that would be calculated by applying the *clustered tree matching* between them.

3.2.1 Motivations

Several motivations lead us to bring these improvements. For example, considering common characteristics shown by Web pages, provides some useful tips: usually, rich sub-levels (i.e. sub-levels with several nodes) represent list items, table rows, menu, etc., more frequently affected by modifications than other elements of Web pages; moreover, analyzing which kind of modifications usually affect Web pages suggests to assign less importance to slight changes happening in deep sub-levels of the DOM tree, this because these are commonly related to missing/added details to elements, etc.

On the one hand, *simple tree matching* ignores these important aspects, on the other *clustered tree matching* exploits information like position and number of mismatches to produce a more accurate result.

3.2.2 Advantages and limitations

The main advantage of our *clustered tree matching* is its capability to calculate an absolute measure of similarity, while *simple tree matching* produces the mapping value between two trees. Moreover, the more the structure of considered trees is complex and similar, the more the measure of similarity established by this algorithm will be accurate. It fits particularly well to matching up HTML Web pages, this because they often own rich and variegated structures.

One important limitation of algorithms based on the tree matching is that they can not match permutations of nodes. Intuitively, this happens because of the dynamic programming technique used to face the problem with computational efficiency; both the algorithms execute recursive calls, scanning sub-trees in a sequential manner, so as to reduce the number of required iterations (e.g. in Figure 1, permutation of nodes [c,b] in A with [b,c] in B is not computed). It is possible to modify the algorithm introducing the analysis of permutations of sub-trees, but this would heavily affect performances. Despite this intrinsic limit, this technique appears to fit very well to our purpose of measuring HTML trees similarity.

It is important to remark that, applying *simple tree matching* to compare simple and quite different trees will produce a more accurate result. Despite that, because of the most of modifications in Web pages are usually slight changes, *clustered tree matching* is far and away the best algorithm to be adopted in building automatically adaptable wrappers. Moreover, this algorithm makes it possible to establish a custom level of accuracy of the process of matching, defining a similarity threshold required to match two trees.

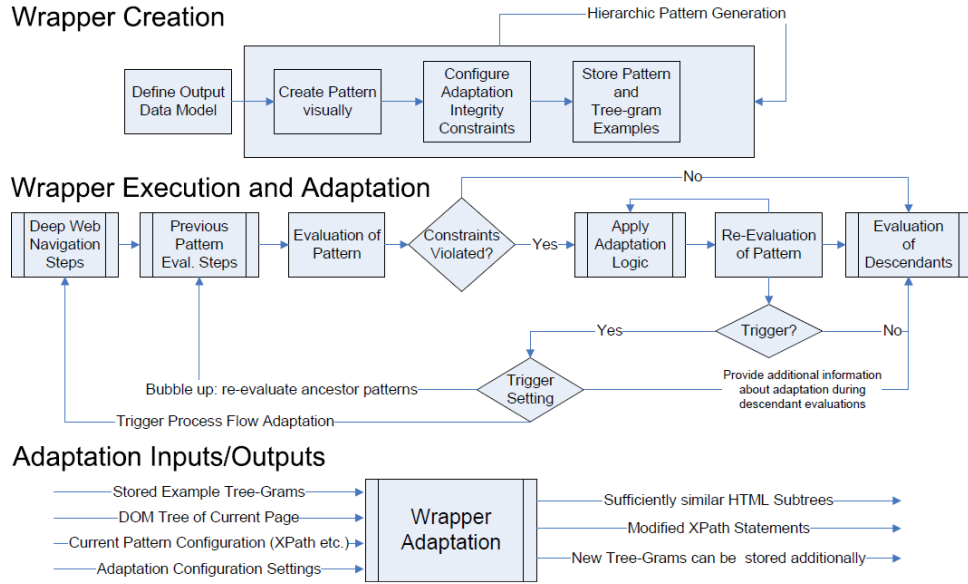


Figure 2: State diagram of Web wrappers design and adaptation in the Lixto Visual Developer.

4 ADAPTABLE WEB WRAPPERS

Based on the adaptation algorithms described above, a proof-of-concept extension to the Lixto Visual Developer (VD) has been implemented. Wrappers are automatically adapted based on given configuration settings, integrity constraints, and triggers.

Usually, wrapper generation in VD is a hierarchical top-down process, e.g. first, a “hotel record” is characterized, and inside the hotel record, entities such as “rating” and “room types”. Such entities are referred to as *patterns*. To define a pattern, the wrapper designer visually selects an example and together with system suggestions generalizes the rule configuration until the desired instances are matched.

In this extension, to support the automatic adaptation process during runtime, the wrapper designer further specifies what it means that extraction failed. In general, this means wrong or missing data, and with integrity constraints one can give indications how correct results look like. Typical *integrity constraints* are:

- *Occurrence restrictions*: e.g. minimum and/or maximum number of allowed occurrence of a pattern instance, minimum and/or maximum number of child pattern instances;
- *Data types*: e.g. all results of a “price” pattern need to be of data type integer.

Integrity constraints can be specified with each pattern individually or be based on a data model (in our case, a given XML Schema). In case integrity constraints are violated during runtime, the adaptation

process for this particular pattern is started.

During wrapper creation, the application designer provides a number of configuration settings to this process. This includes:

- Threshold values;
- Priorities/order of adaptation algorithms used;
- Flags of the chosen algorithm (e.g. using HTML element name as node label, using id/class attributes as node labels, etc.);
- Triggers for bottom-up, top-down and process flow adaptation bubbling;
- Whether stored tree-grams and XPath statements are updated based on adaptation results to be additionally used as inputs in future adaptation procedures (reflecting and addressing regular slight changes of a Web page over time).

Used algorithms for adaptations rely on two inputs (stored example tree-gram(s), DOM tree of current page) and provide as output sub-trees that are sufficiently similar to the original (example) ones, and in consequence a generated XPath statement that matches the nodes (Fig. 2 summarizes the process from design time and execution time perspective).

Algorithms under consideration include the clustered tree matching discussed above, as well as tree-based variants of the Bigram (Collins, 1996) and Jaro-Winkler similarity (Winkler, 1999) (which are of advantage when one assumes that permutations in the tree nodes are likely over time). Moreover, for extraction of leaf nodes which exhibit no inherent tree

structure, we rely on string similarity metrics. Finally, triggers in adaptation settings can be used to force adaptation of further fragments of the wrapper:

- Top-down: forcing adaptation of all/some descendant patterns (e.g. adapt the “price” pattern as well to identify prices within a record if the “record” pattern was adapted).
- Bottom-up: forcing adaptation of a parent pattern in case adaptation of a particular pattern was not successful. Experimental evaluation pointed out that in such cases it is often the problem that the parent pattern already provides wrong or missing results (even if matched by the integrity constraints) and has to be adapted first.
- Process flow: it might happen that particular patterns are no longer detected because the wrapper evaluates on the wrong page. Hence, there is the need to use variations in the deep web navigation processes. A simple approach explored at this time is to use a switch window or back step action to check if the previous window or another tab/pop-up provides the required information.

5 EXPERIMENTAL RESULTS

The best way of measuring reliability of automatically adaptable wrappers is to test their behavior in real world use-cases. Several common areas of application of Web wrappers have been identified: social networks and bookmarking, retail market and comparison shopping, Web search and information distribution, and, finally, Web communities. For each of these fields, we designed a test using a representative Website, studying a total of 7 use-cases, defining wrappers applied to 70 Web pages. Websites like Facebook, Google News, Ebay, etc. are usually subjected to countless, although often invisible, structural modifications; thus, altering the correct behavior of Web wrappers. Table 1 summarizes results: each wrapper automatically tries to adapt itself using both the algorithms described in Section 3. Column referred as *thresh.* means the threshold value of similarity required to match two elements. Columns *tp*, *fp* and *fn* represent true and false positive, and false negative, measures usually adopted to evaluate precision and recall of these kind of tasks.

Performances obtained using the *simple* and the *clustered* tree matching are, respectively, good and excellent; *clustered tree matching* definitely is a viable solution to automatically adapt wrappers with a high degree of reliability (F-Measure greater than 98%). This system provides also the possibility of improv-

		Simple T. M.			Clustered T. M.		
		Precision/Recall			Precision/Recall		
Scenario	thresh.	tp	fp	fn	tp	fp	fn
Delicious	40%	100	4	-	100	-	-
Ebay	85%	200	12	-	196	-	4
Facebook	65%	240	72	-	240	12	-
Google news	90%	604	-	52	644	-	12
Google.com	80%	100	-	60	136	-	24
Kelkoo	40%	60	4	-	58	-	2
Techcrunch	85%	52	-	28	80	-	-
Total	-	1356	92	140	1454	12	42
Recall	-	90.64%			97.19%		
Precision	-	93.65%			99.18%		
F-Measure	-	92.13%			98.18%		

Table 1: Evaluation of the reliability of automatically adaptable wrappers applied to real world scenarios.

ing these results including additional checks on *comparable attributes* (e.g. *id*, *name* or *class*). The role of the required accuracy degree is fundamental; experimental results help to highlight the following considerations: very high values of threshold could result in false negatives (e.g. Google news and Google.com scenarios), while low values could result in false positives (e.g. the Facebook scenario). Our solution exploiting the *clustered tree matching* algorithm, designed by us, helps to reduce wrapper maintenance tasks, keeping in mind that, in cases of deep structural changes, it could be required to manually intervene to fix a specific wrapper, since it is impossible to automatically face all the possible malfunctionings.

6 CONCLUSION

In this paper we described several novel ideas, investigating the possibility of designing smart Web wrappers which automatically react to structural modifications of underlying Web pages and adapting themselves to avoid malfunctionings or corrupting extracted data. After explaining the core algorithms on which this system relies, we shown how to implement this feature in Web wrappers. Finally, we analyzed performances of this system through a rigorous testing of the behavior of automatically adaptable wrappers in real world use-cases.

This work opens new scenarios on wrapper adaptation techniques and is liable to several improvements: first of all, avoiding some limitations of the matching algorithms, for example the inability of handling permutations on nodes previously explained, with computationally efficient solutions could be important to improve the robustness of wrappers. One limitation of adopted tree matching algorithms is also

that they do not work very well if new levels of nodes are added or node levels are removed. We already investigated the possibility of adopting different tree similarity algorithms, working better in such cases. We could try to “generalize” other similarity metrics on strings, such as the n-gram distance and the Jaro-Winkler distance. Implementing these two metrics do not require dynamic programming and might be computationally efficient; in particular, variants of the Bigram distance on trees might work well with permutations of groups of nodes and the Jaro-Winkler distance could better reflect missing or added node levels. Another idea is investigating the possibility of improving matching criteria including additional information to be compared during the tree match up process (e.g. full path information, all attributes, etc.); then, exploiting logic-based rules (e.g. regular expressions, string edit distance, and so on) to analyze textual properties.

Finally, the tree-grammar, already exploited to store a light-weight signature of the structure of elements identified by the wrapper, could be extended for classifying topologies of templates frequently shown by Web pages, in order to define *standard protocols* of automatic adaptation in these particular contexts. Adaptation in the deep web navigation is a different topic than adaptation on a particular page, but also extremely important for wrapper adaptation. Future work will comprise to investigate focused spidering techniques: instead of explicit modeling of a work flow on a Web page (form fill-out, button clicks, etc.) we develop a tree-grammar based approach that decides for a given Web page which template it matches best and executes the data extraction rules defined for this template. Navigation steps are carried out implicitly by following all links and DOM events that have been defined as interesting, crawling a site in a focused way to find the relevant information.

Concluding, the system of designing automatically adaptable wrappers described in this paper has been proved to be robust and reliable. The *clustered tree matching* algorithm is very extensible and it could be adopted for different tasks, also not strictly related to Web wrappers (e.g. operations that require matching up trees could exploit this algorithm).

REFERENCES

- Baumgartner, R., Gottlob, G., and Herzog, M. (2009). Scalable web data extraction for online market intelligence. *Proc. VLDB Endow.*, 2(2):1512–1523.
- Bille, P. (2005). A survey on tree edit distance and related problems. *Theoretical Computer Science*, 337(1-3):217–239.
- Chidlovskii, B. (2001). Automatic repairing of web wrappers. In *Proc. of the 3rd international workshop on Web information and data management*, pages 24–30.
- Collins, M. J. (1996). A new statistical parser based on bigram lexical dependencies. In *Proc. of the 34th Annual Meeting on Association for Computational Linguistics*, pages 184–191, Morristown, NJ, USA.
- Ferrara, E. and Baumgartner, R. (2011). *Automatic Wrapper Adaptation by Tree Edit Distance Matching*. Combinations of Intelligent Methods and Applications. Springer Verlag.
- Ferrara, E., Fiumara, G., and Baumgartner, R. (2010). Web Data Extraction, Applications and Techniques: A Survey. Technical report.
- Kim, Y., Park, J., Kim, T., and Choi, J. (2008). Web information extraction by HTML tree edit distance matching. In *Convergence Information Technology, 2007. International Conference on*, pages 2455–2460.
- Kowalkiewicz, M., Kaczmarek, T., and Abramowicz, W. (2006). MyPortal: robust extraction and aggregation of web content. In *Proc. of the 32nd international conference on Very large data bases*, pages 1219–1222.
- Laender, A., Ribeiro-Neto, B., Silva, A. D., and JS (2002). A brief survey of web data extraction tools. *ACM Sigmod*, 31(2):84–93.
- Lerman, K., Minton, S., and Knoblock, C. (2003). Wrapper maintenance: A machine learning approach. *Journal of Artificial Intelligence Research*, 18(2003):149–181.
- Meng, X., Hu, D., and Li, C. (2003). Schema-guided wrapper maintenance for web-data extraction. In *Proc. of the 5th ACM international workshop on Web information and data management*, pages 1–8, NY, USA.
- Raposo, J., Pan, A., Álvarez, M., and Viña, A. (2005). Automatic wrapper maintenance for semi-structured web sources using results from previous queries. *SAC '05: Proc. of the 2005 ACM symposium on Applied computing*, pages 654–659.
- Selkow, S. (1977). The tree-to-tree editing problem. *Information Processing Letters*, 6(6):184 – 186.
- Tai, K. (1979). The tree-to-tree correction problem. *Journal of the ACM (JACM)*, 26(3):433.
- Winkler, W. E. (1999). The state of record linkage and current research problems. Technical report, Statistical Research Division, U.S. Census Bureau.
- Wong, T. and Lam, W. (2005). A probabilistic approach for adapting information extraction wrappers and discovering new attributes. In *ICDM'04. Proc. of the fourth IEEE International Conference on Data Mining*, pages 257–264.
- Yang, W. (1991). Identifying syntactic differences between two programs. *Software - Practice and Experience*, 21(7):739–755.
- Zhai, Y. and Liu, B. (2005). Web data extraction based on partial tree alignment. In *WWW '05: Proc. of the 14th International Conference on World Wide Web*, pages 76–85, New York, NY, USA.